

Final Report
NASA Contract #NCC-5085

3D Structured Grid Adaptation

D. W. Banks and M. M. Hafez
Department of Mechanical and Aeronautical Engineering
University of California, Davis
Davis, CA. 95616

September, 1996

1 Introduction

Grid adaptation for structured meshes is the art of using information from an already existing, but poorly resolved, solution to automatically redistribute the grid points in such a way as to improve the resolution in regions of high error and thus the quality of the solution. A rough outline of the idea is:

1. Generate a grid via some standard algorithm.
2. Calculate a solution on this grid.
3. Adapt the grid to this solution.
4. Recalculate the solution on this adapted grid.
5. Repeat steps 3 and 4 to satisfaction.

steps 3 and 4 can be repeated until some 'optimal' grid is converged to but typically this is not worth the effort and just two or three repeat calculations are necessary. They may also be repeated every 5-10 time steps for unsteady calculations. The point of going through all this effort, when one could just use steps 1 and 2 with a very fine mesh, is to reduce the total time necessary to obtain a solution of a given accuracy. This entails having an automated way of redistributing the grid points to refine the solution where there is a larger error. However, care must be taken not to draw away too many points from other regions or to distort the grid to such an extent that excessive errors are introduced into the simulation.

Section 2 details the methods used in the current research to carry out the redistribution of mesh. First though, some explanation of grid quality issues, error estimation and previously researched point redistribution algorithms is given.

Since the number of variables associated with the location of each grid point can be cumbersome, some clarification is given. Each grid point in a structured grid has, by definition, indices (i, j, k) that correspond to the point's location in the data structure. Each point (i, j, k) has physical coordinates (x, y, z) . It also has computational coordinates (ξ, η, ζ) which are identical to its (i, j, k) indices and these values are kept constant as the grid evolves. An alternate set of computational coordinates (ξ', η', ζ') will evolve just as (x, y, z) do relative to the original computational coordinates. The variables s^i, s^j, s^k and will be used for arc-lengths in physical space along the grid lines of constant i, j, k , respectively. Similarly, t^i, t^j, t^k will be used for the arc lengths in (ξ', η', ζ') space. $\vec{r}_{i,j,k}$ will be used as the location of point (i, j, k) in physical space.

1.1 Grid Qualities

Grid generation (step 1 from above) is an attempt to produce an 'optimal,' or hopefully at least useful mesh. However, this is done without any detailed *a priori* knowledge of the solution so the quality of the mesh have to be measured by generic qualities such as smoothness, quantified by the Laplacian of the three computational coordinates

$$\Delta\xi, \Delta\eta, \text{ and } \Delta\zeta$$

or the related, 1D parameters of stretching in the three coordinate directions

$$\sigma_\xi = \frac{s_{i+1,j,k}^i - s_{i,j,k}^i}{s_{i,j,k}^i - s_{i-1,j,k}^i}$$

$$\sigma_\eta = \frac{s_{i,j+1,k}^j - s_{i,j,k}^j}{s_{i,j,k}^j - s_{i,j-1,k}^j}$$

$$\sigma_\zeta = \frac{s_{i,j,k+1}^k - s_{i,j,k}^k}{s_{i,j,k}^k - s_{i,j,k-1}^k}$$

Another quality which is often optimized in grid generation is orthogonality or, inversely, skewness. Orthogonality is directional in nature and can be defined in multiple ways. One possible definition would be the orthogonality between the three computational coordinate directions

$$\nu_{\xi\eta} = \left(\frac{\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}}{|\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}|} \right) \cdot \left(\frac{\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|} \right)$$

$$\nu_{\xi\zeta} = \left(\frac{\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}}{|\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}|} \right) \cdot \left(\frac{\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}}{|\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}|} \right)$$

$$\nu_{\eta\zeta} = \left(\frac{\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|} \right) \cdot \left(\frac{\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}}{|\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}|} \right)$$

or possibly in a one-sided sense (as is more often the case in grid generation)

$$\nu_{\xi-\eta} = \left(\frac{\vec{r}_{i,j,k} - \vec{r}_{i-1,j,k}}{|\vec{r}_{i,j,k} - \vec{r}_{i-1,j,k}|} \right) \cdot \left(\frac{\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|} \right)$$

$$\nu_{\xi+\eta} = \left(\frac{\vec{r}_{i,j,k} - \vec{r}_{i+1,j,k}}{|\vec{r}_{i,j,k} - \vec{r}_{i+1,j,k}|} \right) \cdot \left(\frac{\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|} \right)$$

... etc

These measures were chosen due to their impact on the quality of the solution.

Other grid qualities that may effect the solution but rarely used directly in the grid generation process are straightness, quantified by

$$\mu_\xi = \left(\frac{\vec{r}_{i+1,j,k} - \vec{r}_{i,j,k}}{|\vec{r}_{i+1,j,k} - \vec{r}_{i,j,k}|} \right) \times \left(\frac{\vec{r}_{i,j,k} - \vec{r}_{i-1,j,k}}{|\vec{r}_{i,j,k} - \vec{r}_{i-1,j,k}|} \right)$$

$$\mu_\eta = \left(\frac{\vec{r}_{i,j+1,k} - \vec{r}_{i,j,k}}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j,k}|} \right) \times \left(\frac{\vec{r}_{i,j,k} - \vec{r}_{i,j-1,k}}{|\vec{r}_{i,j,k} - \vec{r}_{i,j-1,k}|} \right)$$

$$\mu_\zeta = \left(\frac{\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k}}{|\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k}|} \right) \times \left(\frac{\vec{r}_{i,j,k} - \vec{r}_{i,j,k-1}}{|\vec{r}_{i,j,k} - \vec{r}_{i,j,k-1}|} \right)$$

and aspect ratio

$$AR_\xi = \frac{|\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}|}{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|}$$

$$AR_\eta = \frac{|\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}|}{|\vec{r}_{i+1,j,k} - \vec{r}_{i-1,j,k}|}$$

$$AR_\zeta = \frac{|\vec{r}_{i,j+1,k} - \vec{r}_{i,j-1,k}|}{|\vec{r}_{i,j,k+1} - \vec{r}_{i,j,k-1}|}$$

Of course, some pre-solution adaptation is possible through mechanisms such as stretched and high aspect ratio grids to capture boundary layers and/or wakes and clustering about curved surfaces and where shocks are expected to occur. However, a more automated method is needed to truly reduce the solution time.

adaptation to the solution is accomplished by calculating a local error indicator at each grid node and then regenerating, or refining the grid to cluster points about the high error regions. Regeneration is what is typically used with structured meshes while refinement is useful, by definition, only with unstructured or overset meshes.

1.2 Error Estimation

Traditionally a standard combination of the first and second derivative of a specified flow variable f has been used as the error indicator of choice in structured grid methods.

$$\omega_i = \alpha \frac{\delta_i f}{\delta_i s_i} + \beta \frac{\Delta_i f / \Delta_i s_i - \nabla_i f / \nabla_i s_i}{\delta_i s_i / 2}.$$

f could be any function of the flow solution. Typically, density, pressure, Mach number, or velocity magnitude are used and for many cases have been successful. However, there are two main problems with this sort of indicator. First, It is not very generic and will inevitably cause a user with a complicated flow field some real headaches. Second, it has a hard time distinguishing between strong and weak features and will therefore have difficulty refining flows with shocks of different strengths which occurs quite often.

As an alternative to a physically based indicators, there is the possibility of using something based a bit more theoretically on the numerics of a computational scheme. The error E in a numerical solution is typically expressed as

$$E = Ch^p$$

where C is some unknown value, h is the cell size, and p is the order of the numerical scheme.

Since h and p are known for a given grid and scheme the error in the solution must be estimated by making a guess at C which is a function of the solution and grid metrics other than cell size. Traditional estimates (like the one given above) make an attempt to find C from some sort of "generic" analysis of what is common to most schemes.

One way to get a more precise estimate of C is to look to the modified equations (assuming a finite difference method is being used) If we were to do the full modified equation analysis of a first order upwind scheme being applied to a scalar advection law

$$\frac{F_{i,j} - F_{i-1,j}}{\Delta \xi} + \frac{G_{i,j} - G_{i,j-1}}{\Delta \eta} = 0$$

Where F and G are the co-variant flux components of the Euler equations in the ξ and η directions, respectively.

Taylor series expansion tells us that this is actually representative of the equation

$$F_\xi + G_\eta + \frac{\Delta \xi}{2} F_{\xi\xi} + \frac{\Delta \eta}{2} G_{\eta\eta} + O(\Delta^2) = 0$$

From this one can derive an expression for the local error that includes some zero-order terms that can be eliminated by the use of appropriate metrics terms.

$$\frac{1}{J} (y_{\eta\xi} - y_{\xi\eta}) f + \frac{1}{J} (x_{\xi\eta} - x_{\eta\xi}) g + \frac{1}{2J} (y_{\eta\xi\xi} - y_{\xi\eta\eta}) f + \frac{1}{2J} (x_{\xi\eta\eta} - x_{\eta\xi\xi}) g$$

Here, f and g are the flux terms in the x and y directions and J is the Jacobian of the coordinate transformation.

One also gets some terms that are best controlled by maintaining good grid qualities such as stretching and straightness.

$$\begin{aligned} & \frac{1}{2J} (2y_{\eta\xi}x_{\xi} - 2y_{\xi\eta}x_{\eta} + y_{\eta}x_{\xi\xi} - y_{\xi}x_{\eta\eta}) f_x \\ & + \frac{1}{2J} (2y_{\eta\xi}y_{\xi} - 2y_{\xi\eta}y_{\eta} + y_{\eta}y_{\xi\xi} - y_{\xi}y_{\eta\eta}) f_y \\ & + \frac{1}{2J} (2x_{\xi\eta}x_{\xi} - 2x_{\eta\xi}x_{\eta} - x_{\eta}x_{\xi\xi} + x_{\xi}x_{\eta\eta}) g_x \\ & + \frac{1}{2J} (2x_{\xi\eta}y_{\xi} - 2x_{\eta\xi}y_{\eta} - x_{\eta}y_{\xi\xi} + x_{\xi}y_{\eta\eta}) g_y \end{aligned}$$

and some terms that can be reduced by the use of adaptive gridding

$$\begin{aligned} & \frac{1}{2J} (y_{\eta}x_{\xi}^2 - y_{\xi}x_{\eta}^2) f_{xx} \\ & \frac{1}{J} (2x_{\xi}y_{\xi}y_{\eta} - 2x_{\eta}y_{\eta}y_{\xi}) f_{xy} \\ & \frac{1}{2J} (y_{\eta}y_{\xi}^2 - y_{\xi}y_{\eta}^2) f_{yy} \\ & \frac{1}{2J} (x_{\xi}x_{\eta}^2 - x_{\eta}x_{\xi}^2) g_{xx} \\ & \frac{1}{J} (2x_{\xi}x_{\eta}y_{\eta} - 2x_{\eta}x_{\xi}y_{\xi}) g_{xy} \\ & \frac{1}{2J} (x_{\xi}y_{\eta}^2 - x_{\eta}y_{\xi}^2) g_{yy} \end{aligned}$$

It becomes obvious very quickly that this is not a very practical approach to error indicators. The method is cumbersome and extremely scheme dependent. A stand-alone grid adaptation routine would have to have separate indicators for any flow solver that called it. However, it does provide us with confirmation that the error is directly related to metric terms and the first and second derivatives of the solution.

Other, more generic indicators derived, for the most part, by researchers of unstructured methods can be obtained by testing out the solution dependence on h or p . If the grid is coarsened in one or more directions and the solution is integrated across the elements in the course and fine meshes, the difference should provide a good indicator of where further refinement should give better results. Similarly, one could compare the solutions obtained with different shape functions (e.g. the use of quadratic shape functions over two neighboring linear cells versus the addition of their linear contributions. These sorts of indicators hopefully could provide a useful alternative to the smooth indicators we are using now.

1.3 Grid Movement Schemes

Solution adaptive remeshing for structured grids has been around for more than fifteen years now. Several methods have been developed, most of which are related in one way or another to the concept of equidistribution of an error indicator, ω , either over the lines of constant ξ , η , and/or ζ or over the volumes of the grid. The basic premise of equidistribution schemes is that the optimal solution (i.e. that with the least error) will be obtained on a grid where the local error multiplied by some measure of the cell size is constant for the entire grid. In one dimension this is summarized as

$$\omega_{i+1/2} (s_i - s_{i-1}) = C \quad \text{for all } i$$

In more dimensions, one can apply this equation in an alternating direction fashion, independently distributing the error along the lengths of each computational coordinate direction.

$$\begin{aligned} \omega_{i+1/2} (s_i - s_{i-1}) &= C_i \quad \text{for all } i \\ \omega_{j+1/2} (s_j - s_{j-1}) &= C_j \quad \text{for all } j \\ \omega_{k+1/2} (s_k - s_{k-1}) &= C_k \quad \text{for all } k \end{aligned}$$

Alternatively, the error can be distributed using

$$\omega_{i,j,k} / J_{i,j,k} = C \quad \text{for all } (i, j, k)$$

where J is the Jacobian of the transformation and thus the inverse of the volume of the grid cells.

In general, whether one is using a multiple 1D adaptation scheme or a volume based method, a majority of methods can be grouped into one of three categories:

- variational or optimization schemes,
- methods that modify the original grid generation equations via source terms
- force analogy methods.

No matter which of these methods is used, when one begins to move grid points around for the purpose of adaptation, there also must be some constraint placed on point movement such that grid qualities like stretching/smoothing, orthogonality, and straightness will not be compromised by the error equidistribution.

Variational grid adaptation was pioneered by Brackbill and Saltzman [1], who constructed a functional combining error equidistribution with grid smoothness and orthogonality. The method showed some promise but was criticized for being computationally expensive. Since then, other variational methods have arisen and gained popularity such as those employed by Jacquotte and Cabello [2] who's methods have been used on both structured and unstructured grids for adaptation and quality enhancement. In [3] Eiseman extended the 1D variational error equidistribution scheme to multiple dimensions by repeatedly applying it along grid lines in alternating directions and called it alternating direction adaptation.

Efforts to modify the elliptic grid generation equations have enjoyed a thorough investigation over the years. The scheme starts out with the Euler-Lagrange equations for smoothness and then adds source terms to these equations to obtain other desired qualities including solution adaptivity. The adaptivity source terms were investigated independently by Dwyer [4], Anderson [5], and Thompson [6] The latter was implemented in the popular production code adaptive EAGLE. Another notable effort came from Roache *et al* [7] who developed the notion of a reference grid

which has been modified for use with the method presented in this proposal. The elliptic equations did not get all the attention. Klopfer [8], attempted to adjust the volume source terms in the hyperbolic grid generation equations to obtain an adapted mesh. However the efforts seem to have stopped there.

Force analogous methods attempt to provide an intuitive way to relocate the grid points by using some sort of attractive force to pull points into the high error regions. In [9] Gnoffo introduced a multidimensional spring analogy to equidistribution. Later, Nakahashi and Deiwert modified this concept to be used in a multiple 1D sense in their self adaptive grid method (SAGE) [10, 11, 12] which sweeps through the grid employing a parabolic-like set of equations in each direction. Information on orthogonality and straightness are maintained by torsion springs linking the currently adapted line to the two neighboring lines that have been adapted previously.

Another force analogy algorithm is that of Benson and McRea [13] which uses a moment analogy to equidistribute the error in computational space. By operating in computational space this scheme has the advantage of easily locating the moving points on the original grid for interpolation purposes by use of a nearest integer operation on the new point locations.

The method proposed in this effort attempts to combine the better qualities of the preceding algorithms into an elliptic spring analogy applied in computational space. Previously, in [14] The authors implemented a similar scheme applied in physical space but have found the present method to be much more robust.

2 Methods

This section will describe in some detail how the Solution Adaptive Remeshing Algorithm (SARA) defines and then moves points into the high error regions. Attention is focused on the aspects of SARA that are unique to this research. However, many of the idea implemented in the scheme could be easily modified to fit the needs of other codes that are currently in use.

2.1 Spring system

the equations used to relocate grid points in SARA are based on a spring analogy applied in computational space. Each point is connected to its six neighbors, as is illustrated in figure 1, by springs with constants k that are equal to the error indicator, ω , of the solution multiplied by a metric term.

$$\begin{aligned}
 k_{i-1/2} &= \omega_{i-1/2} \frac{s_{i,j,k}^i - s_{i-1,j,k}^k}{t_{i,j,k}^i - t_{i-1,j,k}^k} & k_{i+1/2} &= \omega_{i+1/2} \frac{s_{i+1,j,k}^i - s_{i,j,k}^k}{t_{i+1,j,k}^i - t_{i,j,k}^k} \\
 k_{j-1/2} &= \omega_{j-1/2} \frac{s_{i,j,k}^j - s_{i,j-1,k}^k}{t_{i,j,k}^j - t_{i,j-1,k}^k} & k_{j+1/2} &= \omega_{j+1/2} \frac{s_{i,j+1,k}^j - s_{i,j,k}^k}{t_{i,j+1,k}^j - t_{i,j,k}^k} \\
 k_{k-1/2} &= \omega_{k-1/2} \frac{s_{i,j,k}^k - s_{i,j,k-1}^k}{t_{i,j,k}^k - t_{i,j,k-1}^k} & k_{k+1/2} &= \omega_{k+1/2} \frac{s_{i,j,k+1}^k - s_{i,j,k}^k}{t_{i,j,k+1}^k - t_{i,j,k}^k}
 \end{aligned}$$

The springs pull with a force equal to the spring constant times the difference in ξ , η or ζ between the two points. The metric multipliers are needed to relate the forces in computational space to the forces in physical space so that they are analogous to an error equidistribution scheme.

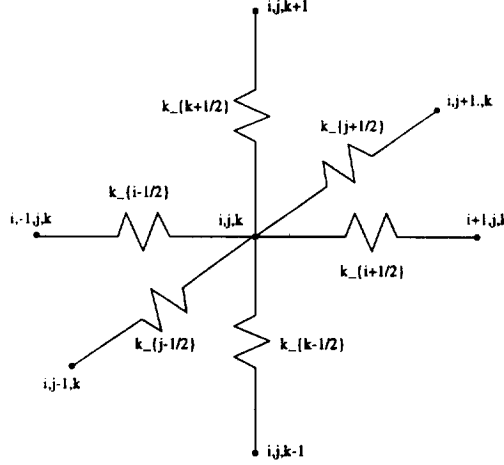


Figure 1: A sketch of the spring system used to relocate grid points in computational space.

This setup results in three force balance equations which are solved for the new locations of each point in the computational domain.

$$\begin{aligned}
 & k_{i-1/2} \nabla_i \xi'_{i,j,k} + k_{i+1/2} \Delta_i \xi'_{i,j,k} + \\
 & k_{j-1/2} \nabla_j \xi'_{i,j,k} + k_{j+1/2} \Delta_j \xi'_{i,j,k} + \\
 & k_{k-1/2} \nabla_k \xi'_{i,j,k} + k_{k+1/2} \Delta_k \xi'_{i,j,k} = 0
 \end{aligned}$$

$$\begin{aligned}
 & k_{i-1/2} \nabla_i \eta'_{i,j,k} + k_{i+1/2} \Delta_i \eta'_{i,j,k} + \\
 & k_{j-1/2} \nabla_j \eta'_{i,j,k} + k_{j+1/2} \Delta_j \eta'_{i,j,k} + \\
 & k_{k-1/2} \nabla_k \eta'_{i,j,k} + k_{k+1/2} \Delta_k \eta'_{i,j,k} = 0
 \end{aligned}$$

$$\begin{aligned}
 & k_{i-1/2} \nabla_i \zeta'_{i,j,k} + k_{i+1/2} \Delta_i \zeta'_{i,j,k} + \\
 & k_{j-1/2} \nabla_j \zeta'_{i,j,k} + k_{j+1/2} \Delta_j \zeta'_{i,j,k} + \\
 & k_{k-1/2} \nabla_k \zeta'_{i,j,k} + k_{k+1/2} \Delta_k \zeta'_{i,j,k} = 0
 \end{aligned}$$

Where the convention of using Δ and ∇ for forward and backward differences, respectively, has been used. Note that because the spring constants are a function of grid point location so these equations are highly non-linear and must be solved for iteratively.

The choice of adapting in computational space was made for two reasons, first this makes the re-interpolation of the solution and error indicator on the the new adapted grid much easier. If the adaptation was in physical space, then before this interpolation, SARA would have to search for where each new point lay in the old grid via some sort of stencil walking scheme as was used in [14]. However, when computational coordinates are used, the nearest point can be found with a simple nearest integer function. This is extremely convenient, never fails, and has a negligible cost. The second reason, which was the motivating factor for making the switch is that adapting high aspect ratio grids near curved boundaries is very difficult in physical space and can result in grid lines crossing into the body unless expensive countermeasures are implemented.

Grid quality maintenance in SARA is extremely simple. If one assumes that the original grid has desirable qualities of orthogonality and straightness, then to maintain these qualities in, for example, the i direction one can constrain the first and second difference of ξ in the j and k directions. This will insure that grid points along lines of constant j and k will remain fairly constant and straight in the computational domain which will, in turn, preserve the qualities of orthogonality and straightness in physical space. To implement this constraint, one need only increase the values of the spring constants in the j and k directions when solving the equation for ξ . Equal values of these cross springs on opposite sides of a node will maintain straightness while a bias to one side or the other will maintain orthogonality in that direction.

If we take the original equations for grid point movement and rename the cross-springs τ with their new purpose in mind we come up with the following equations for grid point relocation.

$$\begin{aligned} k_{i-1/2} \nabla_i \xi'_{i,j,k} + k_{i+1/2} \Delta_i \xi'_{i,j,k} &+ \\ \tau_{j-1/2} \nabla_i \xi'_{i,j,k} + \tau_{j+1/2} \Delta_i \xi'_{i,j,k} &+ \\ \tau_{k-1/2} \nabla_i \xi'_{i,j,k} + \tau_{k+1/2} \Delta_i \xi'_{i,j,k} &= 0 \end{aligned}$$

$$\begin{aligned} \tau_{i-1/2} \nabla_i \eta'_{i,j,k} + \tau_{i+1/2} \Delta_i \eta'_{i,j,k} &+ \\ k_{j-1/2} \nabla_i \eta'_{i,j,k} + k_{j+1/2} \Delta_i \eta'_{i,j,k} &+ \\ \tau_{k-1/2} \nabla_i \eta'_{i,j,k} + \tau_{k+1/2} \Delta_i \eta'_{i,j,k} &= 0 \end{aligned}$$

$$\begin{aligned} \tau_{i-1/2} \nabla_i \zeta'_{i,j,k} + \tau_{i+1/2} \Delta_i \zeta'_{i,j,k} &+ \\ \tau_{j-1/2} \nabla_i \zeta'_{i,j,k} + \tau_{j+1/2} \Delta_i \zeta'_{i,j,k} &+ \\ k_{k-1/2} \nabla_i \zeta'_{i,j,k} + k_{k+1/2} \Delta_i \zeta'_{i,j,k} &= 0 \end{aligned}$$

The τ terms can be manipulated by the user to provide different levels of orthogonality and straightness in different regions of the grid. As it is now, the code uses a user specified constant value for τ in each computational direction for straightness maintenance and another user specified constant at each boundary for orthogonality maintenance. These boundary values for τ are then blended into the interior of the grid.

Another quality controlled by SARA is stretching. Since stretching is also the means by which the algorithm adapts the mesh, we need only find a way of guaranteeing that the grid is not overly stretched for the sake of adaptation. Normally this is done by the application of Laplacian smoothing to the error function. However, this method is imprecise and will alter the minimum and maximum value of the error function.

An alternate method can be found by looking to a one-dimensional spring system for which the force balance reduces to

$$\omega_{i+1/2} \Delta_i s = \omega_{i-1/2} \nabla_i s,$$

which can be rearranged in terms of stretching to yield

$$\sigma_i = \frac{\omega_{i-1/2}}{\omega_{i+1/2}}.$$

In the current method the error function is monitored to see if the ratio of neighboring spring constants will violate the maximum allowed stretching specified by the user. This is done with six sweeps through the grid (backward and forward in each direction) to check and adjust the springs.

A feature of SARA which is related to stretching control is reference grid stretching. In the absence of anything to adapt to locally, the user may desire SARA to maintain the original grid's relative spacing. This can be done by setting the springs to a uniform value which will preserve a uniform spacing in computational space, thus insuring that the original grid's values for stretching will be maintained. The reference grid constant can be used to control the overall adaptivity of the run as large values will completely override most adaptation except in the regions of highest error.

Four boundary conditions are built into the code at the moment. Besides a simple fixed point, Dirichlet type boundary condition there are two conditions that allow the slipping of points along the edge of the boundary. One uses a zeroth order extrapolation that acts as a orthogonal slipping condition as long as the original grid was orthogonal. This is done by switching off all the springs except for the one normal to the boundary. The other slipping condition leaves all the springs active and solves the full equations of point movement along the boundary surface. The final boundary condition available is a periodic condition which has been implemented implicitly via a periodic tridiagonal solver.

2.2 Iterative Procedure

The iterative procedure used to solve the equations of point movement is an alternating direction implicit relaxation algorithm. Each iteration includes six planer sweeps through the grid alternating which direction in the plane is implicit and which is vectorized. For each sweep the springs are:

- interpolated from their values on the original grid.
- modified for stretching control.
- multiplied by the current metrics for adaptation in computational space.
- modified to include the reference grid springs.

Each correction is relaxed in order to guarantee convergence.

To help speed up the convergence of the algorithm a grid sequencing scheme was implemented. The idea behind this method as it applies to SARA is to initially adapt on a coarser subset of the grid to get an estimate of where the points should move to. Once this has been accomplished, the remaining points are then interpolated back into the adapted grid with a higher (2nd or 3rd) order method to keep the grid smooth. An illustration of the grid sequencing process is shown in figure 2.

To use this method the user must first specify a subregion of the grid that they want to sequence. They then specify the number of points that they want in each direction and whether they want the grid to be coarsened evenly in computational space (i.e. every second or fourth point) or coarsened evenly according to arc length along the grid lines. This latter option is very helpful for coarsening viscous grids in the normal direction for speedy adaptation along the surface of the geometry.

Use of grid sequencing has two extremely useful benefits. First, it greatly increases the rate at which an adapted grid can be obtained. Second, the amount of memory needed to efficiently run SARA is much smaller when it is working on the subset of the grid. This enables the use of SARA on a common work station even when grids for very large problems are being adapted.

2.3 post processing

Several post process operations have been implemented to help make SARA a bit more flexible and useful for a greater variety of problems. Figure 3 shows a grid at the boundary of an adapted region. The first post process option is a simple extrapolation of the new spacing into the grid

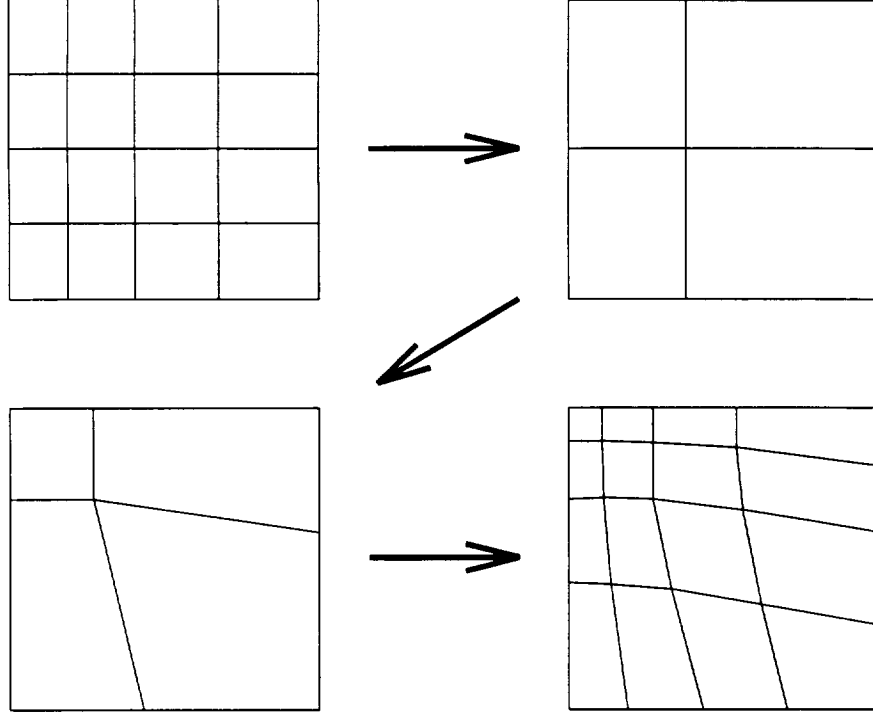


Figure 2: An illustration of the grid sequencing procedure.

from the adapted region. Figure 4 illustrates this method. The extrapolation used can only be 0th order because higher order schemes could easily cause crossed grid lines. The second method is a reflection of the spacing in the adapted region into the grid. This is useful for problems with symmetry planes such as the Onera M6 wing case presented in section 3.2. An illustration of the reflected post process is shown in figure 5.

2.4 Error Indicators

As an alternative to the traditional error indicators outlined in section 1.2 an indicator similar to those used for unstructured grid adaptive refinement was implemented. In particular, the work of Aftosmis [15] has been adapted for use with structured grids in SARA. This refinement criterion attempts to separate out the severe errors in shock regions from those of more smooth features. To flag a shock region the method looks to the vector of normalized undivided second differences of pressure in each direction

$$\overline{(\Delta_i \nabla_i) P} = \frac{P_{i+1,j,k} - 2P_{i,j,k} + P_{i-1,j,k}}{P_{i+1,j,k} + 2P_{i,j,k} + P_{i-1,j,k}},$$

and it's magnitude

$$|\overline{\Delta_i \nabla_i P}| = \sqrt{(\overline{\Delta_i \nabla_i P})^2 + (\overline{\Delta_j \nabla_j P})^2 + (\overline{\Delta_k \nabla_k P})^2}.$$

If this quantity, or any of the components, are greater than a certain tolerance then at least one direction at the point will be flagged as having a shock. Smooth features are picked up with the

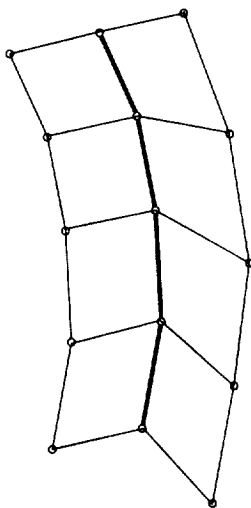


Figure 3: Before extrapolation of adapted spacing.

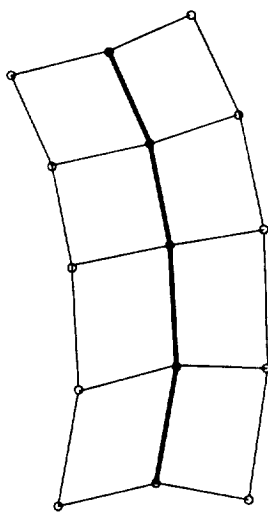


Figure 4: Extension of grid spacing.

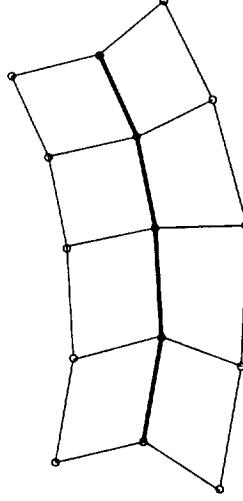


Figure 5: Reflection of grid spacing.

normalized undivided difference of density

$$\overline{\delta_i \rho} = \frac{\rho_{i+1,j,k} - \rho_{i-1,j,k}}{\rho_{i+1,j,k} + \rho_{i-1,j,k}}.$$

This quantity is then normalized in each computational direction to some fraction of the shock point value and added into the overall error function. Viscous features are detected by an undivided second difference of velocity magnitude.

$$\overline{(\Delta_i \nabla_i) V} = \frac{V_{i+1,j,k} - 2V_{i,j,k} + V_{i-1,j,k}}{V_{i+1,j,k} + 2V_{i,j,k} + V_{i-1,j,k}},$$

A sense of directionality is brought into the scheme by only tagging directions that line up within a certain tolerance (set to 20° .)

This scheme produces a very jagged error function which is not a problem for an unstructured grid but for use with structured grids it must be smoothed. However, as was noted in section 2.1 the simple stretching control feature of SARA can be used to smooth out the error indicator which eliminates this problem.

3 Test Problems

3.1 Supersonic Airfoil

A simple parabolic arc airfoil in supersonic flow was used to test the algorithm. The solution was originally obtained on an O type mesh where the airfoil was defined by 198 points in the chord-wise direction with a leading and trailing edge spacings of 0.05% chord. The flow solution at 0 deg angle of attack and $Ma = 1.7$ was obtained by use of the thin layer Navier-Stokes code OVERFLOW [16]. The Original grid and pressure contours are shown in figures 6-7. Figures 8 and 9 show the final grid and solution. The adapted solution is seen to be markedly improved. With the initial grid the leading and trailing shocks dissipated before reaching the far field boundary while the adapted grid solution has no trouble propagating shocks out of the domain.

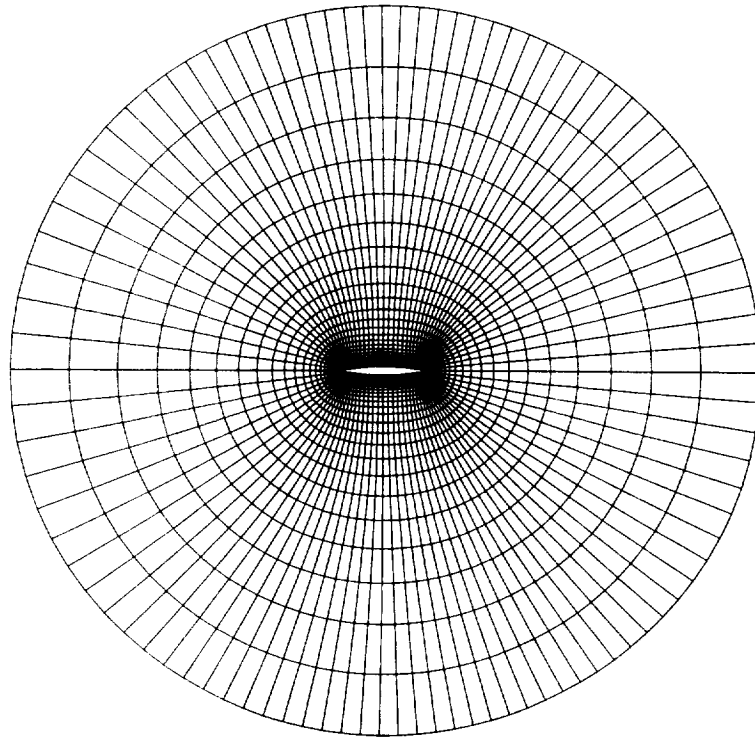


Figure 6: Original mesh for supersonic airfoil.

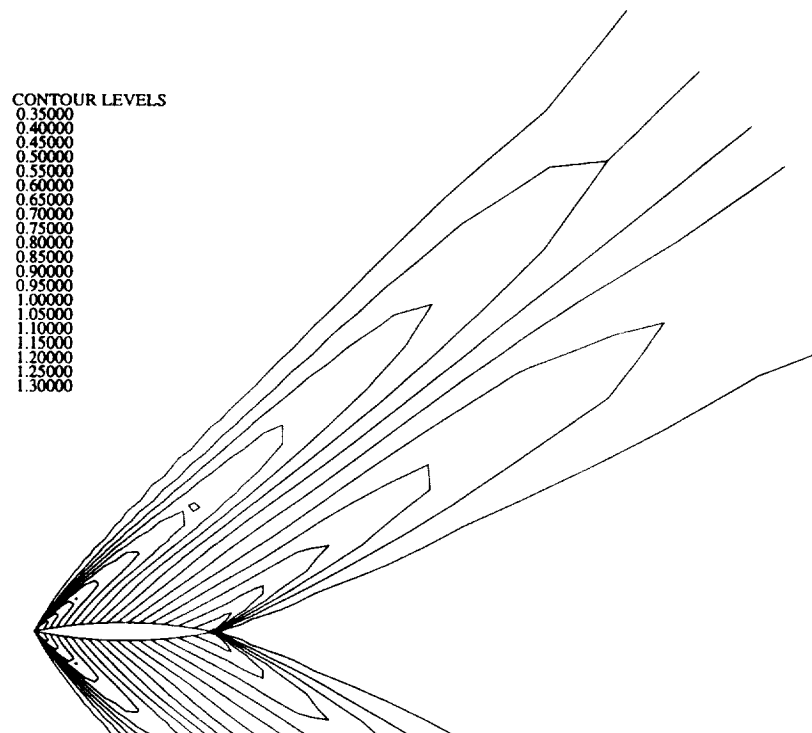


Figure 7: Pressure contours on original grid.

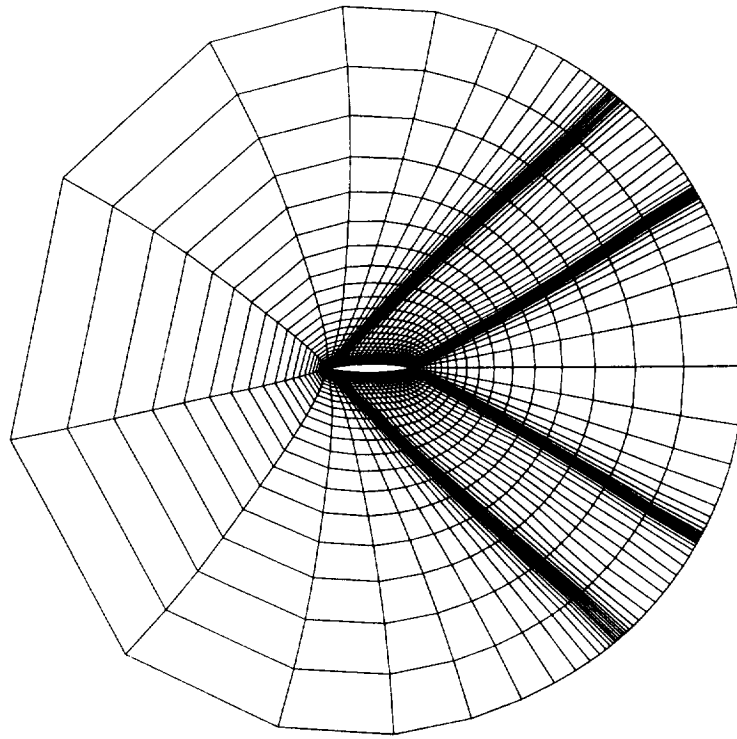


Figure 8: Adapted mesh for the supersonic airfoil.

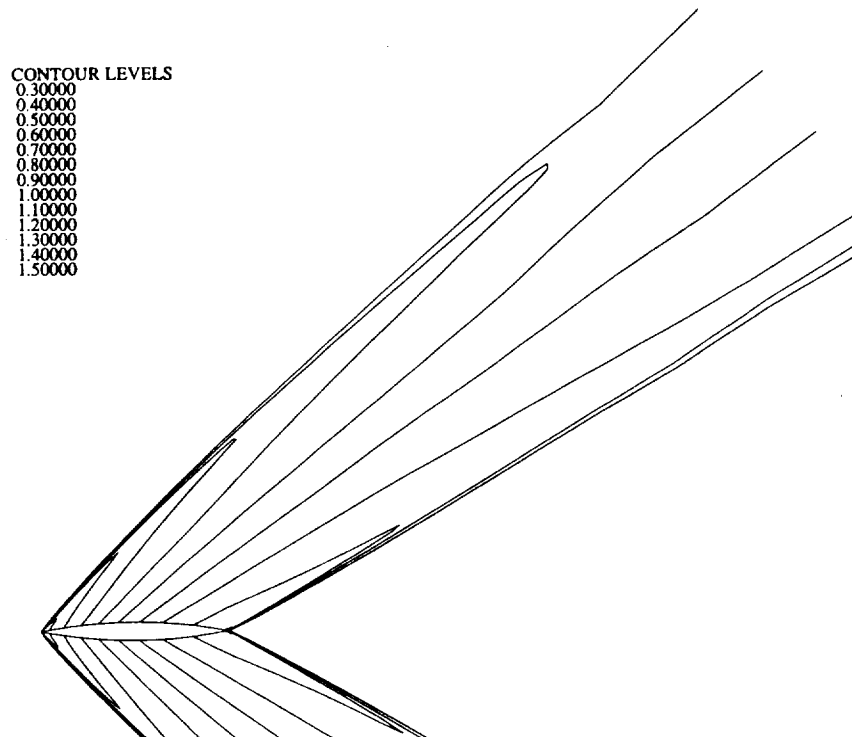


Figure 9: Pressure contours on the adapted mesh.

3.2 Onera M6 wing

For a practical application of the code, high Reynolds number flow at transonic Mach number over an ONERA M6 wing with a rounded tip was attempted. The solution was originally obtained on a 269x35x67 C-O type mesh where the wing was defined by 200 points in the chord-wise direction with a leading edge spacing of 0.1% chord and a trailing edge spacing of 0.2%. The flow solution for an angle of attack of 5 degrees, $Re = 7.6 \times 10^9$, and $Ma = 0.82$ was obtained by use of the thin layer Navier-Stokes code OVERFLOW [16]. The Original grid and solutions are shown in figures 10-14. The grid used for adaptation contained only half of the points in the chord-wise and normal directions to enable the use of SARA interactively on Ames' Cray C90 Eagle. Figures 15 and 18 show the course adapted grid and the solution obtained on it. Note that even though only a quarter of the points were used, the solutions before and after adaptation are qualitatively the same.

4 Conclusions

The grid movement scheme based on a spring analogy in the computational space was shown to work much more robustly for highly curved surfaces at the price of a more complex boundary spacing treatment. The addition of the grid sequencing algorithm enhanced the usefulness of SARA by enabling it to be run on small workstations and at a much lower computational cost. The adaptive criterion constructed by Aftosmis for unstructured refinement was shown to work quite well when modified for use with a structured remeshing scheme. This result should encourage the investigation of other refinement criterion for grid adaptation.

References

- [1] Brackbill J.U. and J.S. Saltzman. Adaptive zoning for singular problems in two dimensions. *JCP*, 46:342-368, 1982.
- [2] O.P. Jacquotte and J. Cabello. Three-dimensional grid generation method based on a variational principle. *A91-22721*, 1991.
- [3] P. Eiseman. Alternating direction adaptive grid generation. *AIAA Journal*, 23:551-560, Apr 1985.
- [4] K Matsung and H. A. Dwyer. Adaptive methods for elliptic grid generation. *JCP*, 77, 1988.
- [5] D. A. Anderson. Constructing adaptive grids with poisson grid generators. In *Numerical Grid Generation in CFD and Related Fields*, pages 125-136, 1986.
- [6] Phu B. Luong, Thompson Joe F., and B. Gatlin. Adaptive eagle: Solution-adaptive and quality enhancing multi-block grids for arbitrary domains. *AIAA 91-1593 CP*.
- [7] P.J. Roache, K. Salari, and S. Steinberg. Hybrid adaptive poisson grid generation and grid smoothness. *Communications in Applied Numerical Methods*, 7:345-354, 1991.
- [8] G. H. Klopfer. Solution adaptive meshes with a hyperbolic grid generator. *Numerical Grid Generation in CFD and Related Fields*, 1987.

- [9] P.A. Gnoffo. A finite-volume, adaptive grid algorithm applied to planetary entry flowfields. *AIAA Journal*, 21:1249–1254, Sep 1983.
- [10] K. Nakahashi and G.S. Deiwert. Self-adaptive grid method with application to airfoil flow. *AIAA Journal*, 25:513–520, Apr 1987.
- [11] M.J. Djomehri and G.S. Deiwert. Three-dimensional self-adaptive grid method for complex flows. In *Numerical Grid Generation in CFD and Related Fields*, pages 277–287, 1988.
- [12] C. Davies and E. Venkatapathy. Application of a solution adaptive grid schemes, sage, to complex three-dimensional flows. *AIAA 91-1594 CP*.
- [13] R.A. Benson and D.S. McRae. A solution-adaptive mesh algorithm for dynamic/static refinement of two and three-dimensional grids. In *Numerical Grid Generation in CFD and Related Fields*, pages 185–199, 1991.
- [14] David W. Banks. A solution-adaptive relaxation algorithm for structured grids. Master’s thesis, U.C. Davis, 1993.
- [15] M. Aftosmis. Viscous flow simulation using and upwind method for hexahedral based adaptive meshes. *AIAA-paper*, (93-0772), Jan. 1994.
- [16] Pieter G. *et al* Buning. Overflow user’s manual. Technical report, NASA Ames, 1992.

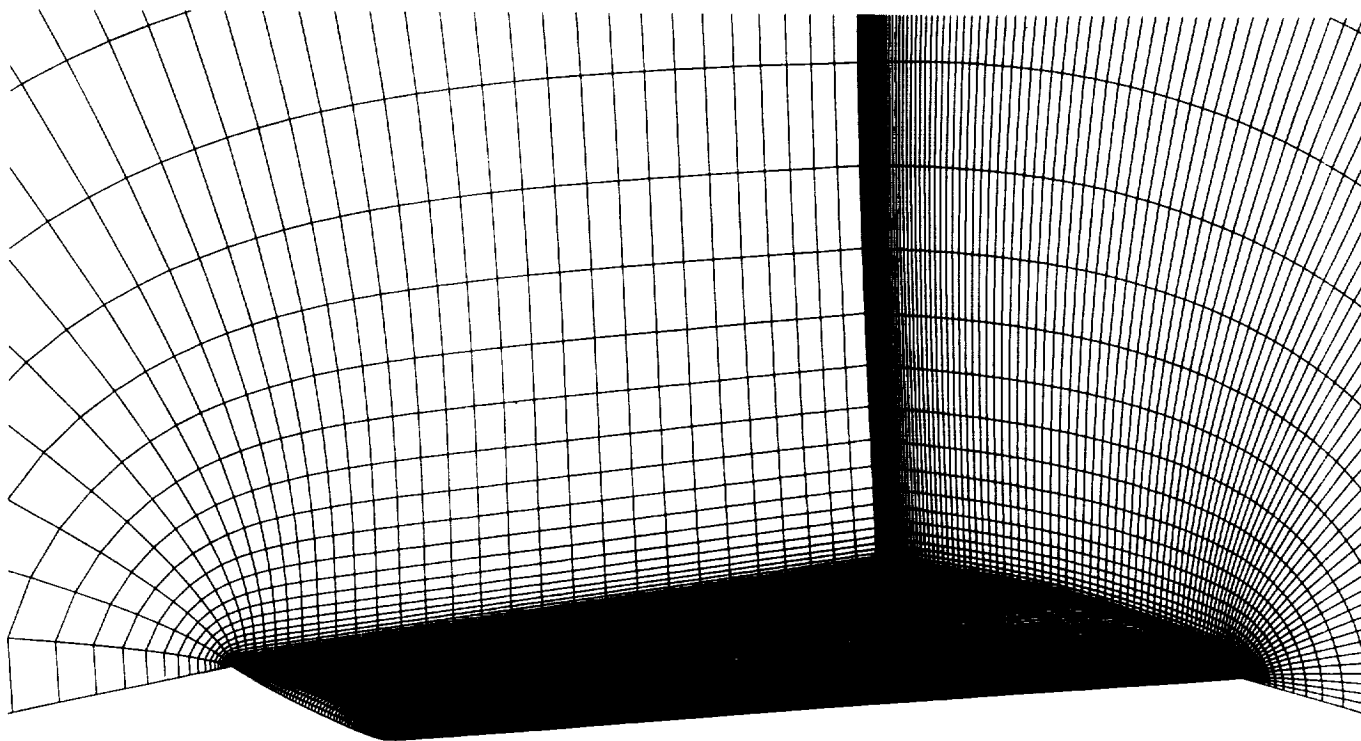


Figure 10: ONERA M6 C-O grid topology

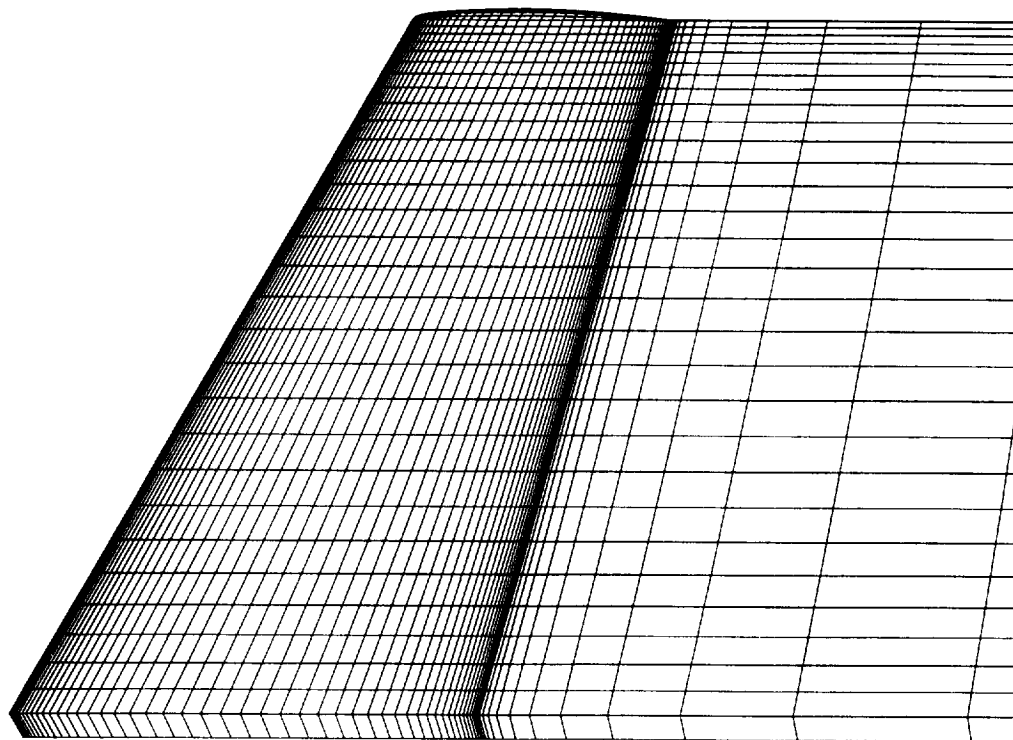


Figure 11: Original grid on the upper surface of the wing.

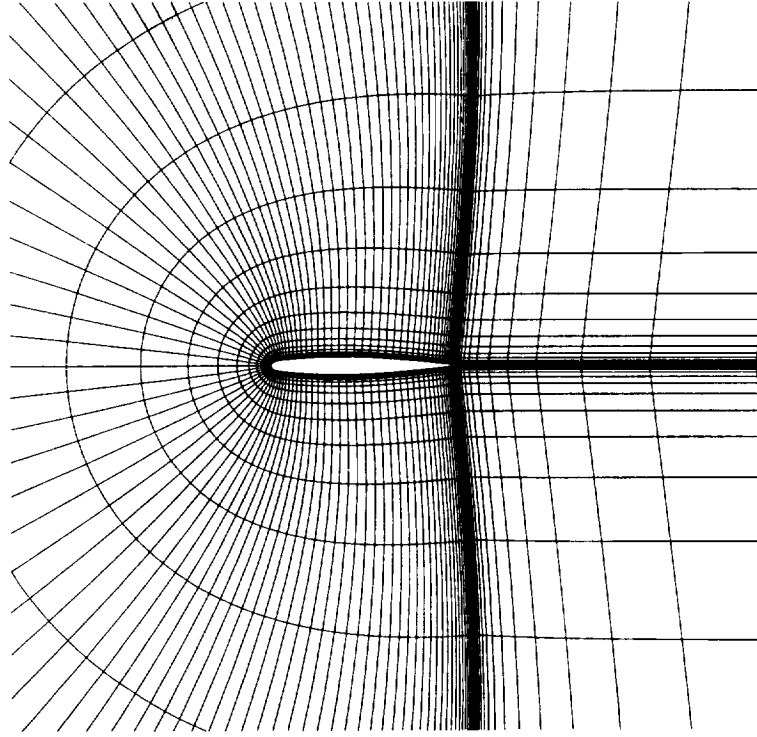


Figure 12: Original grid at the symmetry plane.

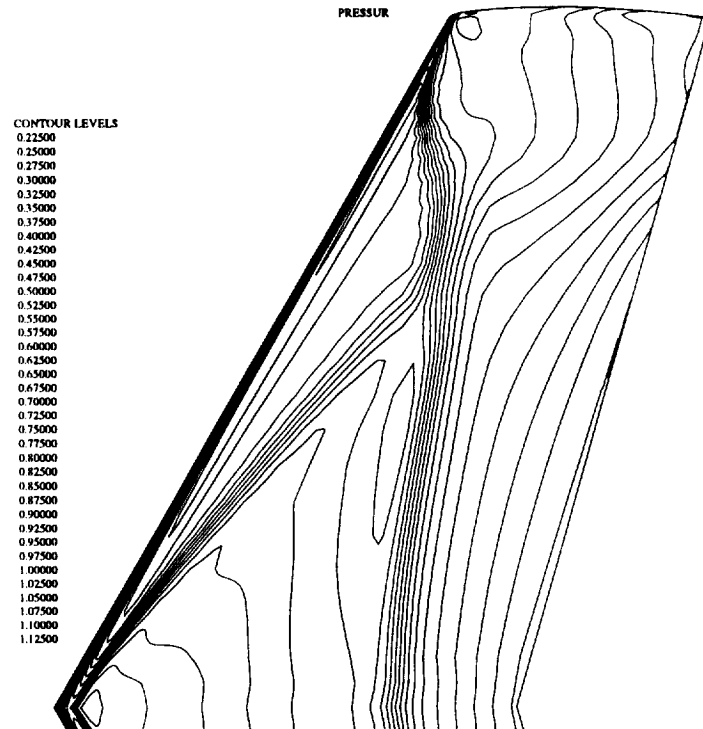


Figure 13: Pressure contours computed on the original grid on the upper wing surface.

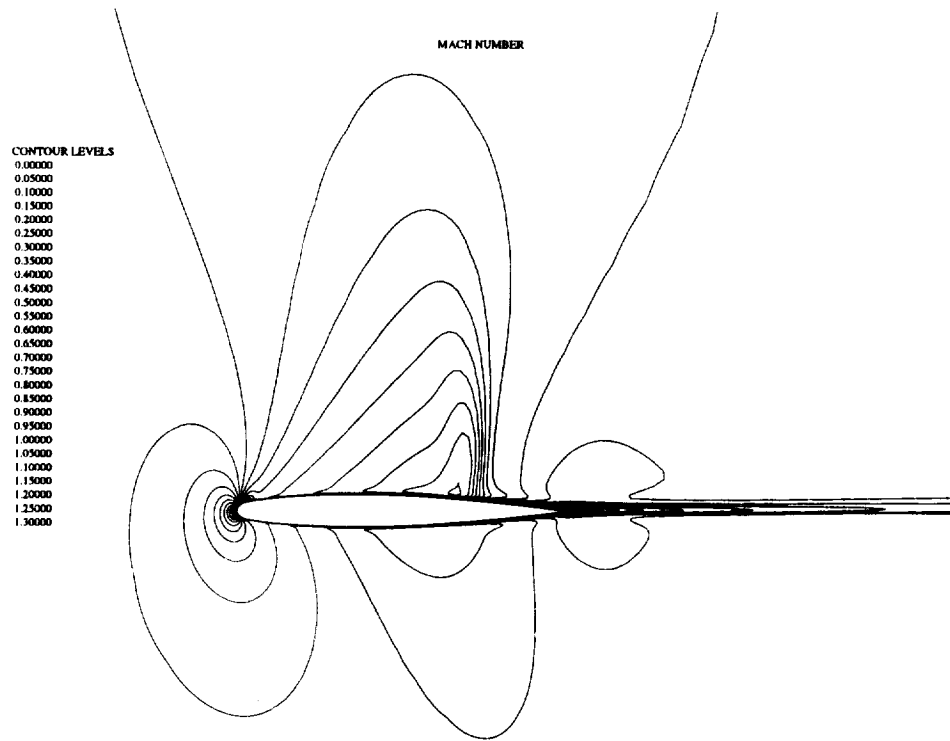


Figure 14: Mach contours computed on the original grid at the symmetry plane.

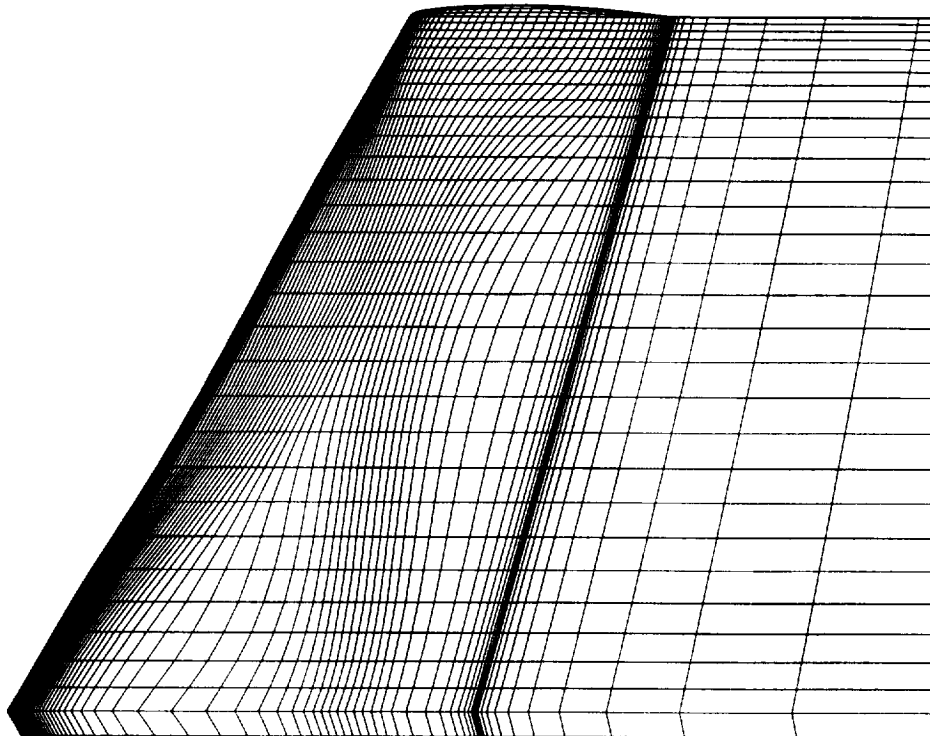


Figure 15: Coarse adapted mesh on the upper surface of the wing.

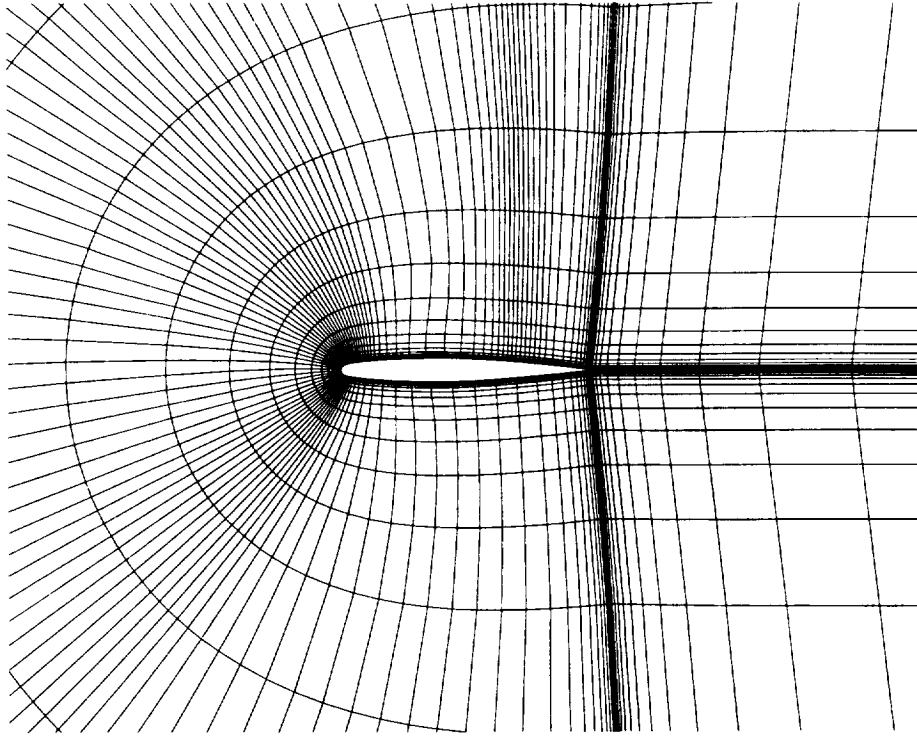


Figure 16: Coarse adapted mesh at the symmetry plane.

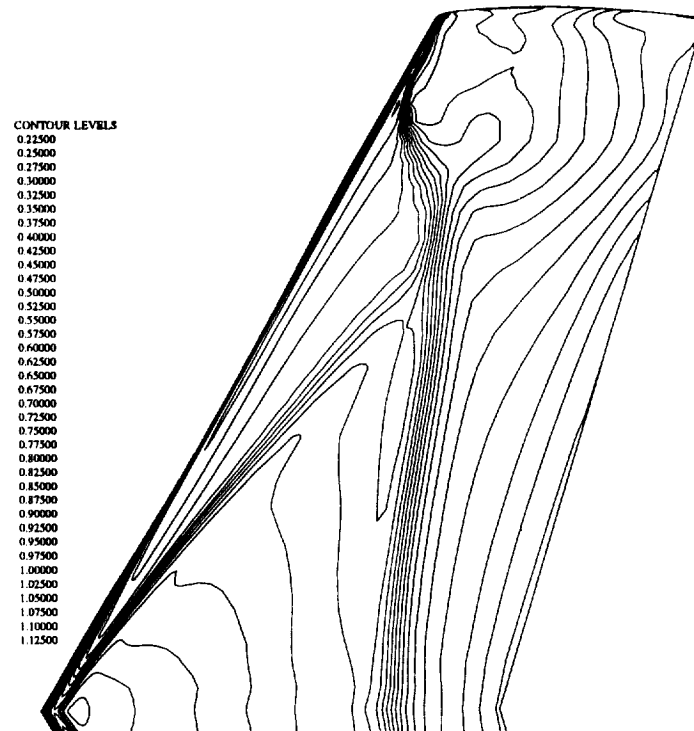


Figure 17: Pressure contours computed on the coarse adapted grid on the upper wing surface.

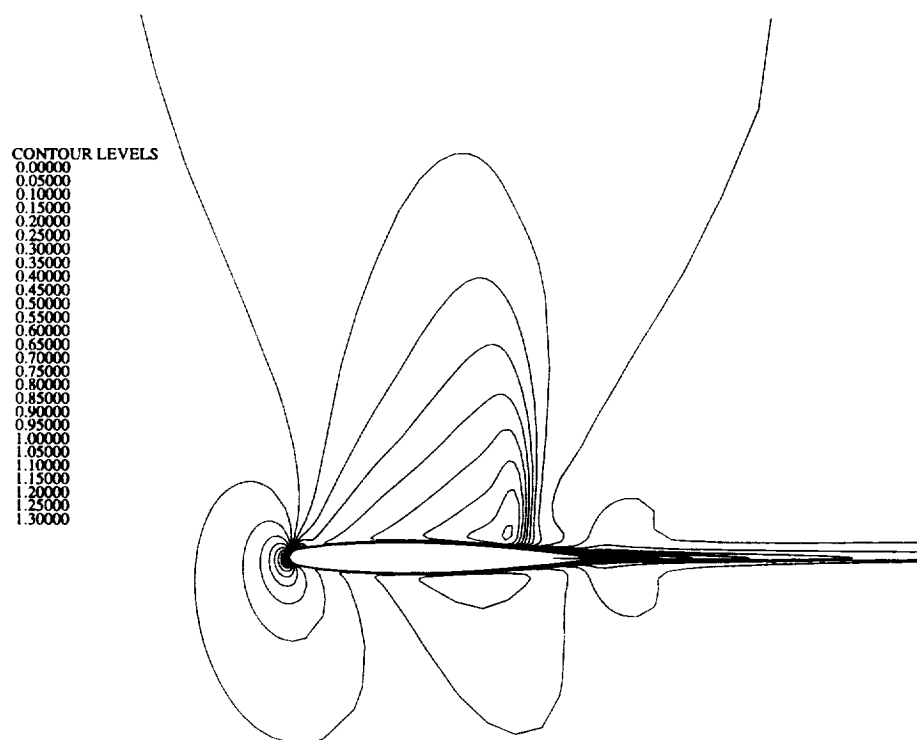


Figure 18: Mach contours computed on the coarse adapted grid at the symmetry plane.